**Problem Name:** Letter Combination of a phone number

**Topics:** Hash Table, String, Backtracking

**Companies:** Amazon, Microsoft, Facebook, Google, Uber, Apple, Bloomberg, Oracle, Swiggy, Adobe, Twitter, Epic Systems, Snapchat, Intuit, Cisco, Nutanix, Goldman Sach, eBay, Twilo, Duolingo, Square, Samsung, Tesla, Capital one, Atlassian, Morgan Stanley, Qualtrics, JPMorgan, ServiceNow, Databricks, Salesforce, VMware, Roblox, American Express, Quora, Dropbox

**Level:** Medium

**Language:** C++

**Problem Statement:** Given a string containing digits from 2-9 inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digit to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

**Input Format:**

First and only line of the input contains string Digits.

Ex:

23

**Output Format:** Print all possible letter combinations that the number could represent

**Constraints:**

* 0 <= digits.length <= 4
* digits[i] is a digit in the range ['2', '9'].

**Examples:**

**Input:** digits = "23"

**Output:** ["ad","ae","af","bd","be","bf","cd","ce","cf"]

**Brute force Solution:**

**Explanation:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<string> letterCombinations(string digits) {

    vector<string> x(10);

    x[2] = "abc";

    x[3] = "def";

    x[4] = "ghi";

    x[5] = "jkl";

    x[6] = "mno";

    x[7] = "pqrs";

    x[8] = "tuv";

    x[9] = "wxyz";

    if(digits.length() == 0){

        return {};

    }

    vector<string> ans;

    if(digits.length() == 1){

            int k = digits[0] - '0';

            string ss = x[k];

            for(int i=0;i<ss.length();i++){

                char c = ss[i];

                string s; s.push\_back(c);

                ans.push\_back(s);

            }

        }else if(digits.length() == 2){

        string s1 = x[digits[0] - '0'];

        string s2 = x[digits[1] - '0'];

        for(int i=0;i<s1.length();i++){

            for(int j=0;j<s2.length();j++){

                string s;

                s.push\_back(s1[i]);s.push\_back(s2[j]);

                ans.push\_back(s);

            }

        }

    }else if(digits.length() == 3){

        string s1 = x[digits[0] - '0'];

        string s2 = x[digits[1] - '0'];

        string s3 = x[digits[2] - '0'];

        for(int i=0;i<s1.length();i++){

            for(int j=0;j<s2.length();j++){

                for(int k=0;k<s3.length();k++){

                string s;

                s.push\_back(s1[i]);s.push\_back(s2[j]);s.push\_back(s3[k]);

                ans.push\_back(s);

                }

            }

        }

    }else{

            string s1 = x[digits[0] - '0'];

        string s2 = x[digits[1] - '0'];

        string s3 = x[digits[2] - '0'];

        string s4 = x[digits[3] - '0'];

        for(int i=0;i<s1.length();i++){

            for(int j=0;j<s2.length();j++){

                for(int k=0;k<s3.length();k++){

                    for(int l=0;l<s4.length();l++)

                    {

                        string s;

                        s.push\_back(s1[i]);

                        s.push\_back(s2[j]);

                        s.push\_back(s3[k]);

                        s.push\_back(s4[l]);

                        ans.push\_back(s);

                    }

                }

            }

        }

    }

    return ans;

}

int main() {

    string digits;

    cin>>digits;

    vector<string> result;

    result = letterCombinations(digits);

    for(int i=0; i<result.size(); i++){

        cout<<result[i]<<" ";

    }

    return 0;

}

**Time Complexity**: O(4n)

**Space Complexity:** O(N)

**Optimized Solution:**

**Explanation:**

Explanation with sample input "123"

Initial state:

* result = {""}

Stage 1 for number "1":

* result has {""}
* candiate is "abc"
* generate three strings "" + "a", ""+"b", ""+"c" and put into tmp,  
  tmp = {"a", "b","c"}
* swap result and tmp (swap does not take memory copy)
* Now result has {"a", "b", "c"}

Stage 2 for number "2":

* result has {"a", "b", "c"}
* candidate is "def"
* generate nine strings and put into tmp,  
  "a" + "d", "a"+"e", "a"+"f",  
  "b" + "d", "b"+"e", "b"+"f",  
  "c" + "d", "c"+"e", "c"+"f"
* so tmp has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }
* swap result and tmp
* Now result has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }

Stage 3 for number "3":

* result has {"ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf" }
* candidate is "ghi"
* generate 27 strings and put into tmp,
* add "g" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
* add "h" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
* add "h" for each of "ad", "ae", "af", "bd", "be", "bf", "cd", "ce", "cf"
* so, tmp has  
  {"adg", "aeg", "afg", "bdg", "beg", "bfg", "cdg", "ceg", "cfg"  
  "adh", "aeh", "afh", "bdh", "beh", "bfh", "cdh", "ceh", "cfh"  
  "adi", "aei", "afi", "bdi", "bei", "bfi", "cdi", "cei", "cfi" }
* swap result and tmp
* Now result has  
  {"adg", "aeg", "afg", "bdg", "beg", "bfg", "cdg", "ceg", "cfg"  
  "adh", "aeh", "afh", "bdh", "beh", "bfh", "cdh", "ceh", "cfh"  
  "adi", "aei", "afi", "bdi", "bei", "bfi", "cdi", "cei", "cfi" }

**Code:**

#include <bits/stdc++.h>

using namespace std;

const vector<string> pad = {

    "", "", "abc", "def", "ghi", "jkl",

    "mno", "pqrs", "tuv", "wxyz"

};

vector<string> letterCombinations(string digits) {

    if (digits.empty()) return {};

    vector<string> result;

    result.push\_back("");

    for(auto digit: digits) {

        vector<string> tmp;

        for(auto candidate: pad[digit - '0']) {

            for(auto s: result) {

                tmp.push\_back(s + candidate);

            }

        }

        result.swap(tmp);

    }

    return result;

}

int main() {

    string digits;

    cin>>digits;

    vector<string> result;

    result = letterCombinations(digits);

    for(int i=0; i<result.size(); i++){

        cout<<result[i]<<" ";

    }

    return 0;

}

**Time Complexity**: O(4^n)

**Space Complexity:** O(n)